Slide 1

Administrivia

- Reminder: Homework 1 due today (by 5pm).

Slide 2

Interprocess Communication

- Processes almost always need to interact with other processes:
  - "Ordering constraints" – e.g., process B uses as input some data produced by process A.
  - Use of shared resources — files, shared memory locations, etc.
- Use of shared resources can lead to "race conditions" — output depends on details of interleaving.
- Processes must communicate to avoid race conditions and otherwise synchronize.
- "Classical IPC problems" — simplified versions of things you often want to do.

Slide 3

Mutual Exclusion Problem

- In many situations, we want only one process at a time to have access to some shared resource.
- Generic/abstract version — multiple processes, each with a "critical region" ("critical section"):

```
while (true) {
    do_cr();        // must be "finite"
    do_non_cr();    // need not be "finite"
}
```

- Goal is to add something to this code such that:
  1. No more than one process at a time can be "in its critical region".
  2. No process not in its critical region can block another process.
  3. No process waits forever to enter its critical region.
  4. No assumptions are made about how many CPUs, their speeds.

Slide 4

Mutual Exclusion Problem, Continued

- We'll look at various solutions (some correct, some not):
  - Using only hardware features always present (some notion of shared variable).
  - Using optional hardware features.
  - Using "synchronization primitives" (abstractions that help solve this and other problems).
- Recall that a correct solution
  - Must work for more than one CPU.
  - Must work even in the face of unpredictable context switches — whatever we're doing, another process can pull the rug out from under us between "atomic operations" (machine instructions).

Slide 5

Sidebar: Atomic Operations

- "Atomic" operation — indivisible, executes without interference from other processes.
- Which of the following are atomic?
  - `x = 1;`
  - `x = x + 1;`
  - `++x;`
  - `if (x == 0) x = 1;`

  (Or does it depend? On what?)

Slide 6

Proposed Solution — Disable Interrupts

- Pseudocode for each process:

```
while (true) {
    disable_interrupts();
    do_cr();
    enable_interrupts();
    do_non_cr();
}
```

- Does it work? reviewing the criteria . . . No.

Slide 7

Proposed Solution — Simple Lock Variable

- Shared variables:

```
int lock = 0;
```

Pseudocode for each process:

```
while (true) {
    while (lock != 0);
    lock = 1;
    do_cr();
    lock = 0;
    do_non_cr();
}
```

- Does it work? reviewing the criteria . . . No.

Slide 8

Proposed Solution — Strict Alternation

- Shared variables:

```
int turn = 0;
```

Pseudocode for process p0:

```
while (true) {
    while (turn != 0);
    do_cr();
    turn = 1;
    do_non_cr();
}
```

Pseudocode for process p1:

```
while (true) {
    while (turn != 1);
    do_cr();
    turn = 0;
    do_non_cr();
}
```

- Does it work? reviewing the criteria . . . No.

Slide 9

Sidebar: Reasoning about Concurrent Algorithms

- For concurrent algorithms (such as various solutions proposed for mutual exclusion problem), testing is less helpful than for sequential algorithms. (Why?)
- May be helpful, then, to try to think through whether they work. How? Idea of "invariant" may be useful:
  - Loosely speaking — "something about the program that's always true". (If this reminds you of "loop invariants" in CSCI 1323 — good.)
  - Goal is to come up with an invariant that's easy to verify by looking at the code and implies the property you want (here, "no more than one process in its critical region at a time").
  - We will do this quite informally, but it can be done much more formally — mathematical "proof of correctness" of the algorithm.

Slide 10

Minute Essay

- Do you (think you) see why the various solutions to the mutual exclusion problem so far work / don't work?
- Give an example (other than those discussed) of a situation in which you think a solution to this problem would be needed.